Unwrapping ADMM: Efficient Distributed Computing via Transpose Reduction

Thomas Goldstein
University of Maryland

Gavin Taylor
US Naval Academy

Kawika Barabin
US Naval Academy

Kent Sayre
US Naval Academy

Abstract

Recent approaches to distributed model fitting rely heavily on consensus ADMM, where each node solves small sub-problems using only local data. We propose iterative methods that solve *global* sub-problems over an entire distributed dataset. This is possible using transpose reduction strategies that allow a single node to solve least-squares over massive datasets without putting all the data in one place. This results in simple iterative methods that avoid the expensive inner loops required for consensus methods. We analyze the convergence rates of the proposed schemes and demonstrate the efficiency of this approach by fitting linear classifiers and sparse linear models to large datasets using a distributed implementation with up to 20,000 cores in far less time than previous approaches.

1 Introduction

We study optimization routines for problems of the form

$$\text{minimize} \quad f(Dx), \tag{1}$$

where $D \in \mathbb{R}^{m\times n}$ is a (large) data matrix and f is a convex function. We are particularly interested in the case that D is stored in a distributed way across N nodes of a network or cluster. In this case, the matrix $D = (D_1^T, D_2^T, \cdots, D_N^T)^T$ is a vertical stack of sub-matrices, each of which is stored on a node. If the function f decomposes across nodes as well, then problem (1) takes the form

$$\text{minimize} \quad \sum_{i\leq N} f_i(D_i x), \tag{2}$$

where the summation is over the N nodes. Problems of this form include logistic regression, support vector machines, lasso, and virtually all generalized linear models [1].

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51.

Most distributed solvers for equation (2), such as ADMM, assume that one cannot solve global optimization problems involving the entire matrix D. Rather, each node alternates between solving small sub-problems involving only local data, and exchanging information with other nodes.

This work considers methods that solve *global* optimization problems over the entire distributed dataset on each iteration using *transpose reduction methods*. Such schemes exploit the following simple observation: when D has many more rows than columns, the matrix D^TD is considerably smaller than D. The availability of D^TD enables a single node to solve least-squares problems involving the entire data matrix D. Furthermore, in many applications it is possible and efficient to compute D^TD in a distributed way. This simple approach solves extremely large optimization problems much faster than the current state-of-the art. We support this conclusion with convergence bounds and experiments involving multi-terabyte datasets.

2 Background

Much recent work on solvers for formulation (2) has focused on the Alternating Direction Method of Multipliers (ADMM) [2, 3, 4], which has become a staple of the distributed computing and image processing literature. The authors of [5] propose using ADMM for distributed model fitting using the "consensus" formulation. Consensus ADMM has additionally been studied for distributed model fitting [6], support vector machines [7], and numerous domain-specific applications [8, 9]. Many variations of ADMM have subsequently been proposed, including specialized variants for decentralized systems [10], asynchronous updates [11, 12], inexact solutions to subproblems [13], and online/stochastic updates [12].

ADMM is a general method for solving the problem

$$\text{minimize} \quad g(x) + h(y), \quad \text{subject to} \quad Ax + By = 0. \tag{3}$$

The ADMM enables each term of problem (3) to be addressed separately. The algorithm in its simplest form begins with estimated solutions x^0, y^0, and a Lagrange mul-

tiplier λ^0. The "scaled" ADMM then generates the iterates

$$\begin{cases} x^{k+1} & = \arg\min_x g(x) + \frac{\tau}{2}\|Ax + By^k + \lambda^k\|^2 \\ y^{k+1} & = \arg\min_y h(y) + \frac{\tau}{2}\|Ax^{k+1} + By + \lambda^k\|^2 \\ \lambda^{k+1} & = \lambda^k + Ax^{k+1} + By^{k+1} \end{cases} \tag{4}$$

where τ is any positive stepsize parameter. Disparate formulations are achieved by different A, B, f, and g. For example, consensus ADMM [5] addresses the problem

$$\text{minimize} \quad \sum_i f_i(x_i), \quad \text{subject to} \quad x_i = y \text{ for all } i \tag{5}$$

which corresponds to (3) with $B = (I, I, \cdots I)^T$, $A = I$, $h(x) = \sum_i f_i(x_i)$, and $g = 0$. Rather than solving a global problem, consensus ADMM performs the parallel updates

$$x_i^{k+1} = \arg\min_{x_i} f_i(x_i) + \frac{\tau}{2}\|x_i - y^k\|^2. \tag{6}$$

The shared variable y is updated by the central server, and Lagrange multipliers $\{\lambda_i\}$ force the $\{x_i\}$ to be progressively more similar on each iteration.

3 Transpose Reduction Made Easy: Regularized Least-Squares

Transpose reduction is most easily understood for regularized least-squares problems; we discuss the general case in Section 4. Consider the problem

$$\text{minimize} \quad J(x) + \frac{1}{2}\|Dx - b\|^2 \tag{7}$$

for some penalty term J. When $J(x) = \mu|x|$ for some scalar μ, this becomes the lasso regression [14]. Typical consensus solvers for problem (7) require each node to compute the solution to equation (6), which is here given by

$$\begin{aligned} x_i^{k+1} &= \arg\min_{x_i} \frac{1}{2}\|D_i x_i - b_i\|^2 + \frac{\tau}{2}\|x_i - y^k\|^2 \\ &= (D_i^T D_i + \tau I)^{-1}(D_i^T b_i + \tau y^k). \end{aligned}$$

During the setup phase for consensus ADMM, each node forms the matrix $D_i^T D_i$, and then computes and caches the inverse (or equivalently the factorization) of $(D_i^T D_i + \tau I)$.

Alternatively, transpose reduction can solve (7) on a *single* machine without moving the entire matrix D to one place, greatly reducing the required amount of computation. Using the simple identity

$$\begin{aligned} \frac{1}{2}\|Dx - b\|^2 &= \frac{1}{2}\langle Dx - b, Dx - b\rangle \\ &= \frac{1}{2}x^T(D^T D)x - x^T D^T b + \frac{1}{2}\|b\|^2 \end{aligned}$$

we can replace problem (7) with the equivalent problem

$$\text{minimize} \quad J(x) + \frac{1}{2}x^T(D^T D)x - x^T D^T b. \tag{8}$$

To solve problem (8), the central server needs only the matrix $D^T D$ and the vector $D^T b$. When D is a "tall" matrix $D \in \mathbb{R}^{m\times n}$, with $n \ll m$, $D^T D$ has only n^2 (rather than nm) entries, small enough to store on a single server. Furthermore, because $D^T D = \sum_i D_i^T D_i$ and $D^T b = \sum_i D_i^T b_i$, the matrices $D^T D$ and $D^T b$ is formed by having each server do computations on its own local D_i, and then reducing the results on a central server.

Once $D^T D$ and $D^T b$ have been computed in the cloud and cached on a central server, the global problem can be solved on this server. This is done using either a single-node ADMM method for small dense lasso (see [5] Section 6.4) or a forward-backward (proximal) splitting method [15]. The latter approach only requires the gradient of $\frac{1}{2}x^T(D^T D)x - x^T D^T b$, which is given by $D^T Dx - D^T b$.

4 Unwrapping ADMM: Transpose Reduction for General Problems

Transpose reduction can be applied to complex problems using ADMM. On each iteration of the proposed method, least-squares problems are solved over the entire distributed dataset. In contrast, consensus methods use subproblems involving only small data subsets.

We aim to solve problem (1) by adapting a common ADMM formulation from the imaging literature [4, 16]. We begin by "unwrapping" the matrix D; we remove it from f using the formulation

$$\text{minimize} \quad f(y), \quad \text{subject to} \quad y = Dx. \tag{9}$$

Applying the ADMM with $A = D$, $B = -I$, $h = f$, and $g = 0$ yields the following iterates:

$$\begin{cases} x^{k+1} & = \arg\min_x \|Dx - y^k + \lambda^k\|^2 = D^+(y^k - \lambda^k) \\ y^{k+1} & = \arg\min_y f(y) + \frac{\tau}{2}\|Dx^{k+1} - y + \lambda^k\|^2 \\ \lambda^{k+1} & = \lambda^k + Dx^{k+1} - y^{k+1}. \end{cases} \tag{10}$$

The x update solves a *global* least squares problem over the entire dataset, which requires the pseudoinverse of D. The y update can be written $y^{k+1} = \text{prox}_f(Dx^{k+1} + \lambda^k, \tau^{-1})$, where we used the *proximal mapping* of f, which is defined as $\text{prox}_f(z, \delta) = \arg\min_y f(y) + \frac{1}{2\delta}\|y - z\|^2$. Provided f is decomposable, the minimization in this update is coordinate-wise decoupled. Each coordinate of y^{k+1} is computed with either an analytical solution, or using a simple 1-dimensional lookup table of solutions.

4.1 Distributed Implementation

While Transpose ADMM is highly effective on a single machine, there are additional benefits in large, distributed datasets. $D = (D_1^T, D_2^T, \cdots, D_N^T)^T$, $y = (y_1^T, y_2^T, \cdots, y_N^T)^T$ and $\lambda = (\lambda_1^T, \lambda_2^T, \cdots, \lambda_N^T)^T$ can all be distributed over N nodes, such that no node has sufficient access to solve the global least squares problem for x^{k+1}. This is where we exploit transpose reduction. The constraint in (9) now becomes $y_i = D_i x$, and the least-squares x update in (10) becomes

$$x^{k+1} = D^+(y^k - \lambda^k) = \left(\sum_i D_i^T D_i\right)^{-1} \sum_i D_i^T (y_i^k - \lambda_i^k). \tag{11}$$

Each vector $D_i(y_i^k - \lambda_i^k)$ can be computed locally on node i. Multiplication by $(\sum_i D_i^T D_i)^{-1}$ (which need only be computed once) takes place on the central server. The distributed method is listed in Algorithm 1. Note the massive dimensionality reduction that takes place when $D^T D = \sum_i D_i^T D_i$ is formed.

Algorithm 1 Transpose Reduction ADMM

1: Central node: Initialize global x^0 and τ
2: All nodes: Initialize local $\{y_i^0\}$, $\{\lambda_i^0\}$
3: All nodes: $W_i = D_i^T D_i, \forall i$
4: Central node: $W = (\sum_i W_i)^{-1}$
5: **while** not converged **do**
6: All nodes: $d_i^k = D_i^T(y_i^k - \lambda_i^k), \forall i$
7: Central Node: $x^{k+1} = W \sum_i d_i^k$
8: All nodes:
$y_i^{k+1} = \arg\min_{y_i} f_i(y_i) + \frac{\tau}{2}\|D_i x^{k+1} - y_i + \lambda_i^k\|^2$
$= \text{prox}_{f_i}(D_i x^{k+1} + \lambda_i^k, \tau^{-1}), \forall i$
9: All nodes: $\lambda_i^{k+1} = \lambda_i^k + D_i x^{k+1} - y_i^{k+1}$
10: **end while**

4.2 Heterogeneous problems

Transpose reduction ADMM is, in a sense, the opposite of consensus. Transpose reduction methods solve a global data-dependent problem on the central node, while the remote nodes only perform proximal operators and matrix multiplications. In contrast, consensus methods solve all data dependent problems in the remote nodes, and no data is ever seen by the central node. This important property makes transpose reduction extremely powerful when data is *heterogeneous* across nodes, as opposed to homogeneous problems where each node's data is drawn from identical distributions. With standard homogeneous Gaussian test problems, all consensus nodes solve nearly identical problems, and thus arrive at a consensus quickly.

In practical applications, data on different nodes often represents data from different sources and is thus not identically distributed. The efficiency of consensus in such (more realistic) situations significantly decreases. Because transpose reduction solves global problems over the entire dataset, the distribution of data over nodes is irrelevant, making these methods insensitive to data heterogeneity. We discuss theoretical reasons for this in Section 6.1, and explore the impact of heterogeneity with synthetic and real data in Section 8.

4.3 Splitting Over Columns

When the matrix D is extremely wide ($m \ll n$), it often happens that each server stores a subset of columns of D rather than rows. Fortunately, such problems can be handled by solving the *dual* of the original problem. The dual of the sparse problem (14) is given by

$$\underset{\alpha}{\text{minimize}} \quad f^*(\alpha) \;\text{ subject to }\; \|D^T\alpha\|_\infty \le \mu \tag{12}$$

where f^* denotes the Fenchel conjugate [17] of f. For example the dual of the lasso problem is simply

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{2}\|\alpha + b\|^2 \;\text{ subject to }\; \|D^T\alpha\|_\infty \le \mu.$$

Problem (12) then reduces to the form (1) with

$$\widehat{D} = \begin{pmatrix} I \\ D^T \end{pmatrix}, \; \hat{f}(z)_k = \begin{cases} \frac{1}{2}\|z_k + b_k\|^2, \text{ for } 1 \le k \le m \\ \mathcal{X}(z_k), \text{ for } k > m \end{cases}$$

where $\mathcal{X}(z)$ is the characteristic function of the ℓ_∞ ball of radius μ. The function $\mathcal{X}(z)$ is infinite when $|z_i| > \mu$ for some i, and zero otherwise. The unwrapped ADMM for this problem requires the formation of $D_i D_i^T$ on each server, rather than $D_i^T D_i$.

5 Applications: Linear Classifiers and Sparsity

In addition to penalized regression problems, transpose reduction can train linear classifiers. If $D \in \mathbb{R}^{m\times n}$ contains feature vectors and $l \in \mathbb{R}^m$ contains binary labels, then a logistic classifier is put in the form (9) by letting $f(z)$ be the logistic loss $f_{lr}(z) = \sum_{k=1}^m \log(1 + \exp(-l_k z_k))$.

Problem (9) also includes support vector machines, in which we solve

$$\text{minimize} \quad \frac{1}{2}\|x\|^2 + Ch(Dx), \tag{13}$$

where C is a regularization parameter, and $h = \sum_{k=1}^M \max\{1 - l_k z_k, 0\}$ is a simple "hinge loss" function. The y update in (10) becomes very easy because the proximal mapping of h has the closed form

$$\text{prox}_h(z, \delta)_k = z_k + l_k \max\{\min\{1 - l_k z_k, \delta\}, 0\}.$$

Note that this algorithm is much simpler than the consensus implementation of SVM, which requires each node to solve

SVM-like sub-problems using expensive iterative methods (see Section A in the supplementary material).

Sparse model fitting problems have the form

$$\text{minimize} \quad \mu|x| + f(Dx) \tag{14}$$

for some regularization parameter $\mu > 0$. Sparse problems can be reduced to the form (1) by defining

$$\widehat{D} = \begin{pmatrix} I \\ D \end{pmatrix}, \qquad \hat{f}(z)_k = \begin{cases} \mu|z_k|, \text{ for } 1 \leq k \leq n \\ f_k(z_k), \text{ for } k > n \end{cases}$$

and then minimizing $\hat{f}(\widehat{D}x)$. Experimental results are presented in Section 8.

6 Convergence Theory

Classical results prove convergence of ADMM but provide no rate [18]. More recently, rates have been obtained by using an unusual measure of convergence involving the change between adjacent iterates [19]. It is still an open problem to directly prove convergence of the *iterates* of ADMM in the general setting. In this section, we take a step in that direction by providing a rate at which the *gradient* of (1) goes to zero, thus directly showing that x^k is a good approximate minimizer of (1). We do this by exploiting the form of transpose reduction ADMM and the analytical tools in [19].

Theorem 1. *If the gradient of f exists and has Lipschitz constant $L(\nabla f)$, then* (10) *shrinks the gradient of the objective function* (1) *with global rate*

$$\begin{aligned}\|\nabla\{f(Dx^k)\}\|^2 =& \|D^T\nabla f(Dx^k)\|^2 \\ \leq& C\frac{\|y^0 - Dx^\star\|^2 + \|\lambda^0 - \lambda^\star\|^2}{k}\end{aligned}$$

where $C = (L(\nabla f) + \tau)^2\rho(D^TD)$ is a constant and $\rho(D^TD)$ denotes the spectral radius of D^TD.

Proof. We begin by writing the optimality condition for the x-update in (10):

$$D^T(Dx^{k+1} - y^k + \lambda^k) = D^T\lambda^{k+1} + D^T(y^{k+1} - y^k) = 0.$$

Note we used the definition $\lambda^{k+1} = \lambda^k + Dx^{k+1} - y^{k+1}$ to simplify (6). Similarly, the optimality condition for the y-update yields

$$\begin{aligned}\nabla f(y^{k+1}) + \tau(y^{k+1} - Dx^{k+1} - \lambda^k) =& \nabla f(y^{k+1}) - \tau\lambda^{k+1} \\ =& 0,\end{aligned}$$

or simply $\nabla f(y^{k+1}) = \tau\lambda^{k+1}$. Combining this with (6) yields $D^T\nabla f(y^k) = \tau D^T(y^k - y^{k+1})$. We now have

$$\begin{aligned}\partial_x\{f(Dx^k)\} =& D^T\nabla f(Dx^k) = D^T\nabla f(y^k + Dx^k - y^k) \\ =& D^T\nabla f(y^k + Dx^k - y^k) \\ & - D^T\nabla f(y^k) + \tau D^T(y^k - y^{k+1})\end{aligned}$$

and so $\|\partial_x\{f(Dx^k)\}\|$ is bounded above by

$$\begin{aligned}&\|D^T\nabla f(y^k + Dx^k - y^k) - D^T\nabla f(y^k)\| + \tau\|D^T(y^k - y^{k+1})\| \\ &\leq L(\nabla f)\|D^T\|_{op}\|Dx^k - y^k\| + \tau\|D^T\|_{op}\|y^k - y^{k+1}\|,\end{aligned}$$

where $\|D^T\|_{op}$ denotes the operator norm of D^T.

We now invoke the following known identity governing the difference between iterates of (4)

$$\begin{aligned}&\|B(y^{k+1} - y^k)\|^2 + \|Ax^{k+1} + By^{k+1}\|^2 \\ &\qquad \leq \frac{\|B(y^0 - y^\star)\|^2 + \|\lambda^0 - \lambda^\star\|^2}{k+1}\end{aligned}$$

(see [19], Assertion 2.10). When adapted to the problem form (1), we obtain

$$\begin{aligned}&\|y^{k+1} - y^k\|^2 + \|Dx^{k+1} - y^{k+1}\|^2 \\ &\qquad \leq \frac{\|y^0 - Dx^\star\|^2 + \|\lambda^0 - \lambda^\star\|^2}{k+1}.\end{aligned} \tag{15}$$

It follows from (15) that both $\|Dx^k - y^k\|$ and $\|y^k - y^{k+1}\|$ are bounded above by $\sqrt{(\|y^0 - Dx^\star\|^2 + \|\lambda^0 - \lambda^\star\|^2)/k}$. Applying this bound to (6) yields

$$\|\partial_x\{f(Dx^k)\}\| \leq \sqrt{C(\|y^0 - Dx^\star\|^2 + \|\lambda^0 - \lambda^\star\|^2)/k}.$$

We obtain the result by squaring this inequality and noting that $\|D\|_{op}^2 = \rho(D^TD)$. □

Note that logistic regression problems satisfy the conditions of Theorem 1 with $L(\nabla f) = 1/4$. Also, better rates are possible using accelerated ADMM [20].

6.1 Linear convergence analysis and heterogeneous data

We now examine conditions under which transpose reduction is guaranteed to have a better worst-case performance bound than consensus methods, especially when data is heterogeneous across nodes. We examine convergence rates for the case of strongly convex f, in which case the iterates of the ADMM (4) are known to converge R-linearly. If $x^\star$ and $\lambda^\star$ are the optimal primal and dual solutions, then

$$\begin{aligned}&\tau\|A(x^{k+1} - x^\star)\|^2 + \tau^{-1}\|\lambda^{k+1} - \lambda^\star\|^2 \\ &\quad \leq (1+\delta)^{-1}\left(\tau\|A(x^{k+1} - x^\star)\|^2 + \tau^{-1}\|\lambda^{k+1} - \lambda^\star\|^2\right)\end{aligned}$$

for some $\delta > 0$ [21]. If we denote the condition number of A by κ_A and the condition numbers of the functions f and g by κ_f and κ_g, then $\delta = \frac{\lambda_{min}(A^TA)}{\lambda_{max}(A^TA)\sqrt{\kappa_g}} = \frac{1}{\kappa_A\sqrt{\kappa_g}}$. Applying this result to consensus ADMM ($A = I$ and $g(x) = \sum_i f_i(D_ix_i)$) where all f_i are identically conditioned, we get

$$\delta_{con} = \frac{\min_i \lambda_{min}(D_i^TD_i)^{\frac{1}{2}}}{\max_i \lambda_{max}(D_i^TD_i)^{\frac{1}{2}}\kappa_f^{\frac{1}{2}}}.$$

On the other hand, transpose reduction removes the matrix D from the objective function and corresponds to (4) with $A = I$ and $g(x) = \sum_i f_i(y_i)$. We thus obtain $\delta_{tr} = 1/\sqrt{\kappa_f}$ for the transpose reduction ADMM. Note that $\delta_{tr} \geq \delta_{con}$, and so the worst case performance of transpose reduction is (significantly) better than consensus. The worst-case linear convergence rate of transpose reduction does not deteriorate for poorly conditioned D because the data term has been moved from the objective into the constraints. If we compare δ_{con} to δ_{tr}, we expect the convergence of consensus methods to suffer with increased numbers of nodes and a more poorly conditioned D.

7 Implementation Details

We compare transpose reduction methods to consensus ADMM using both synthetic and empirical data. We study the transpose reduction scheme for lasso (Section 3) in addition to the unwrapped ADMM (Algorithm 1) for logistic regression and SVM. We built a distributed implementation of both the transpose reduction and consensus optimization methods, and ran all experiments on a large Cray supercomputer hosted by the DOD Supercomputing Resource Center. This allowed us to study experiments ranging in size from very small to extremely large. All distributed methods were implemented using MPI. Stopping conditions for both methods were set using the residuals defined in [5] with $\epsilon_{rel} = 10^{-3}$, and $\epsilon_{abs} = 10^{-6}$.

Many steps were taken to achieve top performance of the consensus optimization routine. The authors of [5] suggest using a stepsize parameter $\tau = 1$; however, better performance is achieved by tuning this parameter. We tuned the stepsize parameter to achieve convergence in a minimal number of iterations on a problem instance with $m = 10,000$ data vectors and $n = 100$ features per vector, and then scaled the stepsize parameter up/down to be proportional to m. It was found that this scaling made the number of iterations nearly invariant to n and m. In the consensus implementation, the iterative solvers for each local logistic regression/SVM problem were warm-started using solutions from the previous iteration.

The logistic regression subproblems were solved using a limited memory BFGS method (with warm start to accelerate performance). The transpose reduced lasso method (Section 3) requires a sparse least-squares method to solve the entire lasso problem on a single node. This was accomplished using the forward-backward splitting implementation FASTA [15, 22].

Note that the consensus solver for SVM requires the solution to sub-problems that cannot be solved by conventional SVM solvers (see (17) in the supplemental material), so we built a custom solver using the same coordinate descent techniques as the well-known solver LIBSVM [23]. By using warm starts and exploiting the structure of the consensus sub-problem, our custom consensus ADMM method solves the problem (17) dramatically faster than standard solvers for problem (13). See Appendix A in the supplementary materials for details.

8 Numerical Experiments

To study transpose reduction in a wide range of settings, we applied consensus and transpose solvers to both synthetic and empirical datasets. We recorded both the *total compute time* and the *wallclock time*. Total compute time is the time computing cores spend performing calculations, excluding communication; wall time includes all calculation and communication.

8.1 Synthetic Data

We study ADMM using both standard homogeneous test problems and the (more realistic) case where data is heterogeneous across nodes.

Lasso problems We use the same synthetic problems used to study consensus ADMM in [5]. The data matrix D is a random Gaussian matrix. The true solution x_{true} contains 10 active features with unit magnitude, and the remaining entries are zero. The ℓ_1 penalty μ is chosen as suggested in [5] — i.e., the penalty is 10% the magnitude of the penalty for which the solution to (7) becomes zero. The observation vector is $b = Dx_{true} + \eta$, where η is a standard Gaussian noise vector with $\sigma = 1$.

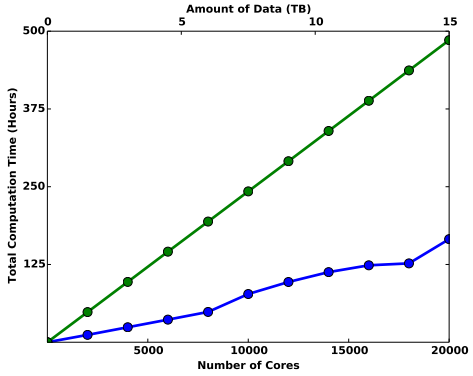
Classification problems We generated a random Gaussian matrix for each class. The first class consists of zero-mean Gaussian entries. The first 5 columns of the second matrix were random Gaussian with mean 1, and the remaining columns were mean zero. The classes generated by this process are not perfectly linearly separable. The ℓ_1 penalty was set using the "10%" rule used in [5].

Heterogeneous Data With homogeneous Gaussian test problems, every node solves nearly identical problems, and we arrive at a consensus quickly. As we saw in the theoretical analysis of Section 6.1, consensus ADMM deteriorates substantially when data is heterogeneous across nodes. To simulate heterogeneity, we chose one random Gaussian scalar for each node, and added it to D_i.

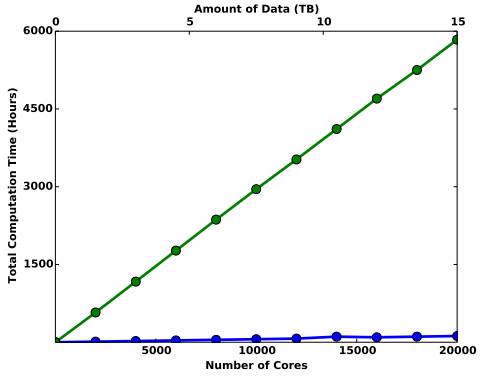
We performed experiments by varying the numbers of cores used, feature vector length, and the number of data points per core. Three representative results are illustrated in Figure 1, while more complete tables appear in supplementary material (Appendix B). In addition, convergence curves for experiments on both homogeneous and heterogeneous data can be seen in Figures 2a and 2b.

8.2 Empirical Case Study: Classifying Guide Stars

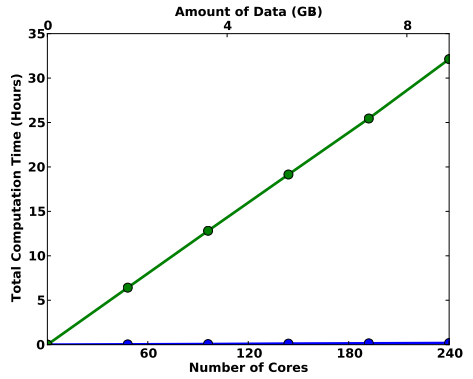
We perform experiments using the Second Generation Guide Star Catalog (GSC-II) [24], an astronomical



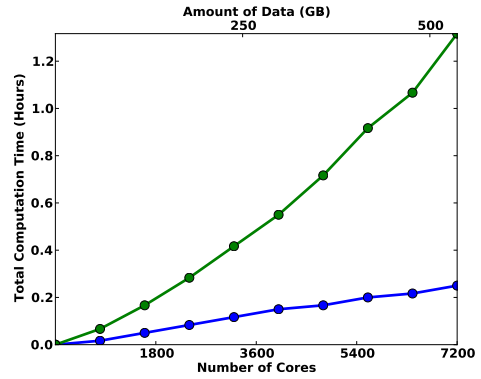
(a) **Logistic regression with homogeneous data**. Experiments used 100,000 data points of 1,000 features each per core.



(b) **Logistic regression with heterogeneous data**. Experiments used 100,000 data points of 1,000 features each per core. Note that transpose ADMM required only 120 hours of compute time for 15 TB of data.



(c) **SVM with homogeneous data**. Experiments used 50,000 data points of 100 features each per computing core.



(d) **Lasso with heterogeneous data**. Experiments used 50,000 data points of 200 features each per computing core.

Figure 1: Selected results from **Consensus ADMM (green)** and **Transpose ADMM (blue)** on three different optimization problems of varying sizes. Every core stores an identically-sized subset of the data, so data corpus size and number of cores are related linearly. The top horizontal axis denotes the total data corpus size, while the bottom horizontal axis denotes the number of computing cores used.

database containing spectral and geometric features for 950 million stars and other objects. The GSC-II also classifies each astronomical body as "star" or "not a star." We train a sparse logistic classifier to discern this classification using only spectral and geometric features.

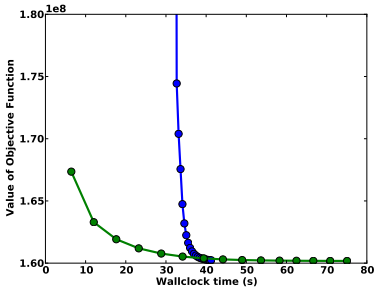
A data matrix was compiled by selecting all spectral and geometric measurements reported in the catalog, and also "interaction features" made of all second-order products. After the addition of a bias feature, the resulting matrix has 307 features per object, and occupies 1.8 TB of space.

Experiments recorded the global objective as a function of wall time. As seen in the convergence curves (Figure 2c) transpose ADMM converged far more quickly than consensus. We also experiment with storing the data matrix across different numbers of nodes. These experiments illustrated that this variable had little effect on the relative performance of the two optimization strategies; transpose methods remained far more efficient regardless of number of cores. See Table 1.

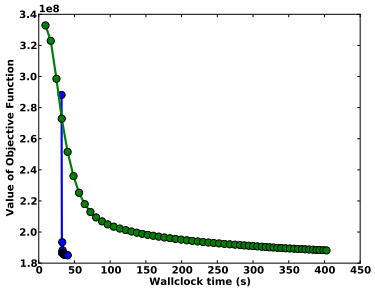
Note the strong advantage of transpose reduction over consensus ADMM in Figure 2c. This confirms the observations of Sections 6.1 and 8.1, where it was observed that transpose reduction methods are particularly powerful for heterogeneous real-world data, as opposed to the identically distributed matrices used in conventional synthetic experiments.

9 Discussion

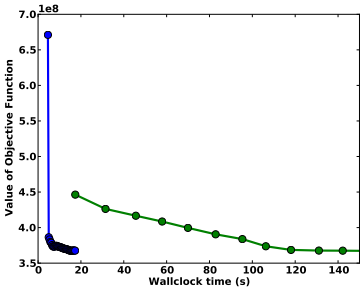
Transpose reduction methods required substantially less computation time than consensus methods in all experiments. As seen in Figure 1, Figure 2, and the supplementary table (Appendix B), performance of both methods scales nearly linearly with the number of cores and amount of data used for classification problems. For lasso (Figure 1d), the runtime of transpose reduction appears to grow sublinearly with the number of cores. This contrasts with consensus methods that have a strong dependence on this parameter. This is largely because the transpose reduction



(a) **Homogeneous data**. Experiments used 7,200 computing cores with 50,000 data points of 2,000 features each.



(b) **Heterogeneous data**. Experiments used 7,200 computing cores with 50,000 data points of 2,000 features each.



(c) **Empirical star data**. Experiments used 2,500 cores to classify 1.8TB of data. Consensus did not terminate until 1160 sec.

Figure 2: Logistic regression objective function vs wallclock time for **Consensus ADMM (green)** and **Transpose ADMM (blue)**.

Cores	T-Wall	T-Comp	C-Wall	C-Comp
2500	0:01:06	11:35:25	0:24:39	31d,19:59:13
3000	0:00:49	12:10:33	0:21:43	32d, 2:44:11
3500	0:00:50	12:17:27	0:17:01	30d, 7:56:19
4000	0:00:45	12:38:24	0:29:53	40d, 13:38:19

Table 1: Wall clock and total compute times for logistic regression on the 1.8TB Guide Star Catalog. "T-" denotes results for transpose reduction and "C-" denotes consensus. Times format is (days,) hours:mins:secs.

method solves all data-dependent problems on a single machine, whereas consensus optimization requires progressively more communication with larger numbers of cores (as predicted in Section 6.1).

Note that transpose reduction methods need more startup time for some problems than consensus methods because the local Gram matrices $D_i^T D_i$ must be sent to the central node, aggregated, and the result inverted; this is not true for the lasso problem, for which consensus solvers must also invert a local Gram matrix on each node, though this at least saves startup communication costs. This startup time is particularly noticeable when overall solve time is short, as in Figure 2b. Note that even for this problem total computation time and wall time was still substantially shorter with transpose reduction than with consensus methods.

9.1 Effect of Heterogeneous Data

When data is heterogeneous across nodes, the nodes have a stronger tendency to "disagree" on the solution, taking longer to reach a consensus. This effect is illustrated by a comparison between Figures 2a and 2b, or Figures 1a and 1b, where consensus methods took much longer to converge on heterogeneous data sets. In contrast, because transpose reduction solves *global* sub-problems across the entire distributed data corpus, it is relatively insensitive to data heterogeneity across nodes. In the same figures, transpose reduction results were similar between the two scenarios, while consensus methods required much more time for the heterogeneous data. This explains the strong advantage of transpose reduction on the GSC-II dataset (Figure 2c, Table 1), which contains empirical data and is thus not uniformly distributed.

9.2 Communication & Computation

Transpose reduction leverages a tradeoff between communication and computation. When N nodes are used with a distributed data matrix $D \in \mathbb{R}^{m\times n}$, each consensus node transmits $x_i \in \mathbb{R}^n$ to the central server, which totals to $O(Nn)$ communication. Transpose reduction requires $O(m)$ communication per iteration, which is often somewhat more. Despite this, transpose reduction is still highly efficient for two reasons. First, consensus requires inner iterations to solve expensive sub-problems, while transpose reduction does not. Second, transpose reduction methods stay synchronized better than consensus ADMM, making communication more efficient on synchronous architectures. The iterative methods used by consensus ADMM for logistic regression and SVM sub-problems do not terminate at the same time on every machine, especially when the data is heterogeneous across nodes. Consensus nodes must block until all nodes become synchronized. In contrast, Algorithm 1 requires the same computations on each server, allowing nodes to stay synchronized naturally.

10 Conclusion

We introduce transpose reduction ADMM — an iterative method that solves model fitting problems using global least-squares subproblems over a distributed dataset. Theoretical convergence rates are superior for the new approach, particularly when data is heterogeneous across nodes. This is illustrated by numerical experiments using synthetic and empirical data, both homogeneous and heterogeneous, which demonstrate that the transpose reduction can be substantially more efficient than consensus methods.

Acknowledgements

This work was supported by the National Science Foundation (#1535902) and the Office of Naval Research (#N00014-15-1-2676 and #N0001415WX01341).

References

[1] James W. Hardin and Joseph Hilbe. *Generalized Linear Models and Extensions*. College Station, Texas: Stata Press, 2001.

[2] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires. *Rev. Française d'Automat. Inf. Recherche Opérationelle*, 9(2):41–76, 1975.

[3] Roland Glowinski and Patrick Le Tallec. *Augmented Lagrangian and Operator-Splitting Methods in Nonlinear Mechanics*. Society for Industrial and Applied Mathematics, Philadephia, PA, 1989.

[4] Tom Goldstein and Stanley Osher. The Split Bregman method for ℓ_1 regularized problems. *SIAM J. Img. Sci.*, 2(2):323–343, April 2009.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 2010.

[6] T. Erseghe, D. Zennaro, E. Dall'Anese, and L. Vangelista. Fast consensus by the alternating direction multipliers method. *Signal Processing, IEEE Transactions on*, 59(11):5523–5537, Nov 2011.

[7] Pedro A. Forero, Alfonso Cano, and Georgios B. Giannakis. Consensus-based distributed support vector machines. *J. Mach. Learn. Res.*, 11:1663–1707, August 2010.

[8] Ruiliang Chen, Jung-Min Park, and Kaigui Bian. Robust distributed spectrum sensing in cognitive radio networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[9] B. He, M. Tao, and X. Yuan. Alternating direction method with gaussian back substitution for separable convex programming. *SIAM Journal on Optimization*, 22(2):313–340, 2012.

[10] J.F.C. Mota, J.M.F. Xavier, P.M.Q. Aguiar, and M. Puschel. D-admm: A communication-efficient distributed algorithm for separable optimization. *Signal Processing, IEEE Transactions on*, 61(10):2718–2723, May 2013.

[11] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1701–1709. JMLR Workshop and Conference Proceedings, 2014.

[12] Hua Ouyang, Niao He, Long Tran, and Alexander G. Gray. Stochastic alternating direction method of multipliers. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 80–88. JMLR Workshop and Conference Proceedings, 2013.

[13] T.-H. Chang, M. Hong, and X. Wang. Multi-agent distributed optimization via inexact consensus admm. *Signal Processing, IEEE Transactions on*, 63(2):482–497, Jan 2015.

[14] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[15] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a FASTA implementation. *arXiv eprint*, abs/1411.3406, 2014.

[16] Tom Goldstein, Xavier Bresson, and Stanley Osher. Geometric applications of the Split Bregman method: Segmentation and surface reconstruction. *J. Sci. Comput.*, 45:272–293, October 2010.

[17] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[18] Jonathan Eckstein and Dimitri P. Bertsekas. On the douglas-rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[19] Bingsheng He and Xiaoming Yuan. On non-ergodic convergence rate of Douglas-Rachford alternating direction method of multipliers. *Optimization Online*, January 2012.

[20] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

[21] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *UCLA CAM technical report, 12-52*, 2012.

[22] Tom Goldstein, Christoph Studer, and Richard Baraniuk. FASTA: A generalized implementation of forward-backward splitting, January 2015. http://arxiv.org/abs/1501.04979.

[23] Chih C. Chang and Chih J. Lin. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), May 2011.

[24] Barry M Lasker, Mario G Lattanzi, Brian J McLean, Beatrice Bucciarelli, Ronald Drimmel, Jorge Garcia, Gretchen Greene, Fabrizia Guglielmetti, Christopher Hanley, George Hawkins, et al. The second-generation guide star catalog: description and properties. *The Astronomical Journal*, 136(2):735, 2008.

Supplementary Material

A SVM Sub-steps for Consensus Optimization

A common formulation of the support vector machine (SVM) solves

$$\text{minimize} \quad \frac{1}{2}\|x\|^2 + Ch(Dx) \tag{16}$$

where C is a regularization parameter, and h is a simple "hinge loss" function given by $h(z) = \sum_{k=1}^{M} \max\{1 - l_k z_k, 0\}$. The proximal mapping of h has the form $\text{prox}_h(z,\delta)_k = z_k + l_k \max\{\min\{1 - l_k z_k, \delta\}, 0\}$. Using this proximal operator, the solution to the y update in (10) is simply $y^{k+1} = \text{prox}_h\left(Dx^{k+1} + \lambda^k, \frac{C}{\tau}\right)$. Note that this algorithm is much simpler than the consensus implementation of SVM, which requires each node to solve the sub-problem

$$\text{minimize} \quad Ch(Dx) + \frac{\tau}{2}\|x - y\|^2. \tag{17}$$

Despite the similarity of this problem to the original SVM (16), this problem form is *not* supported by available SVM solvers such as LIBSVM [23] and others. However, techniques for the classical SVM problem can be easily adapted to solve (17).

A common numerical approach to solving (16) is the attack its dual, which is

$$\underset{\alpha_i \in [0,C]}{\text{minimize}} \quad \frac{1}{2}\|A^T L\alpha\|^2 - \alpha^T \mathbf{1} = \sum_{i,j} \alpha_i \alpha_j l_i l_j A_i A_j^T - \sum_i \alpha_i. \tag{18}$$

Once (18) is solved to obtain $\alpha^\star$, the solution to (13) is simply given by $w^\star = L^T\alpha$. The dual formulation (18) is advantageous because the constraints on α act separately on each coordinate. The dual is therefore solved efficiently by coordinate descent, which is the approach used by the popular solver LIBSVM [23]. This method is particularly powerful when the number of support vectors in the solution is small, in which case most of the entries in α assume the value 0 or C.

In the context of consensus ADMM, we must solve

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + Ch(Aw, l) + \frac{\tau}{2}\|w - z\|^2. \tag{19}$$

Following the classical SVM literature, we dualize this problem to obtain

$$\underset{\alpha_i \in [0,C]}{\text{minimize}} \quad \frac{1}{2}\|A^T L\alpha\|^2 - \alpha^T\left((1+\tau)\mathbf{1} - \tau L z\right). \tag{20}$$

We then solve (20) for $\alpha^\star$, and recover the solution via

$$w^\star = \frac{A^T L\alpha + \tau z}{1+\tau}.$$

We solve (20) using a dual coordinate descent method inspired by [23]. The implementation has $O(M)$ complexity per iteration. Also following [23] we optimize the convergence by updating coordinates with the largest residual (derivative) on each pass.

Because our solver does not need to handle a "bias" variable (in consensus optimization, only the central server treats the bias variables differently from other unknowns), and by using a warm start to accelerate solve time across iterations, our coordinate descent method significantly outperforms even LIBSVM for each sub-problem. On a desktop computer with a Core i5 processor, LIBSVM solves the synthetic data test problem with $m = 100$ datapoints and $n = 200$ features in 3.4 seconds (excluding "setup" time), as opposed to our custom solver which solves each SVM sub-problem for the consensus SVM with the same dimensions (on a single processor) in 0.17 seconds (averaged over all iterations). When $m = 10000$ and $n = 20$, LIBSVM requires over 20 seconds, while the average solve time for the custom solver embedded in the consensus method is only 2.3 seconds.

B Tables of Results

In the following tables, we use these labels:

- N: Number of data points per core
- F: Number of features per data point
- Cores: Number of compute cores used in computation
- Space: Total size of data corpus in GB (truncated at GB)
- TWalltime: Walltime for transpose method (truncated at seconds)
- TCompute: Total computation time for transpose method (truncated at seconds)
- CWalltime: Walltime for consensus method (truncated at seconds)
- CCompute: Total computation time for consensus method (truncated at seconds)

Logistic regression with homogeneous data

N	F	Cores	Space(GB)	TWalltime	TCompute	CWalltime	CCompute
50000	2000	800	596	0:00:53	6:19:14	0:01:36	17:25:18
50000	2000	1600	1192	0:00:58	12:40:24	0:01:51	1 day 10:51:33
50000	2000	2400	1788	0:01:00	19:05:13	0:01:52	2 days 4:21:25
50000	2000	3200	2384	0:01:00	1 day 1:30:18	0:01:41	2 days 21:46:28
50000	2000	4000	2980	0:00:58	1 day 7:58:24	0:01:39	3 days 15:17:51
50000	2000	4800	3576	0:00:58	1 day 14:27:31	0:02:31	4 days 8:49:58
50000	2000	5600	4172	0:01:00	1 day 21:10:38	0:02:13	5 days 2:16:56
50000	2000	6400	4768	0:01:03	2 days 3:46:42	0:02:08	5 days 19:39:40
50000	2000	7200	5364	0:01:21	2 days 10:36:36	0:01:47	6 days 13:12:59
100000	1000	2000	1490	0:02:09	11:50:56	0:01:58	2 days 0:28:08
100000	1000	4000	2980	0:01:32	1 day 0:05:30	0:04:14	4 days 0:58:47
100000	1000	6000	4470	0:01:40	1 day 12:20:57	0:02:00	6 days 1:36:20
100000	1000	8000	5960	0:00:42	2 days 0:42:49	0:03:33	8 days 1:59:14
100000	1000	10000	7450	0:01:01	3 days 5:30:41	0:02:43	10 days 2:30:10
100000	1000	12000	8940	0:01:16	4 days 0:50:36	0:02:54	12 days 2:59:08
100000	1000	14000	10430	0:01:33	4 days 16:42:20	0:05:00	14 days 3:36:58
100000	1000	16000	11920	0:01:18	5 days 3:40:44	0:03:19	16 days 4:11:34
100000	1000	18000	13411	0:01:07	5 days 6:45:44	0:05:29	18 days 4:56:02
100000	1000	20000	14901	0:01:17	6 days 21:44:52	0:03:14	20 days 5:36:16
5000	2000	4800	357	0:00:33	4:04:11	0:00:26	21:01:22
10000	2000	4800	715	0:00:26	7:51:06	0:01:22	1 day 21:24:47
15000	2000	4800	1072	0:00:38	11:23:22	0:01:37	2 days 19:42:30
20000	2000	4800	1430	0:00:42	15:15:01	0:01:30	3 days 19:27:24
25000	2000	4800	1788	0:00:42	18:59:04	0:01:48	4 days 17:24:59
30000	2000	4800	2145	0:00:47	22:53:25	0:02:04	5 days 16:30:28
35000	2000	4800	2503	0:00:57	1 day 2:43:48	0:02:46	6 days 15:10:40
40000	2000	4800	2861	0:00:54	1 day 6:22:51	0:02:42	7 days 14:58:02
45000	2000	4800	3218	0:00:57	1 day 10:05:17	0:03:02	8 days 15:11:42
50000	2000	4800	3576	0:01:02	1 day 14:28:30	0:03:24	9 days 15:51:21
20000	500	4800	357	0:00:05	2:18:21	0:00:35	20:51:20
20000	1000	4800	715	0:00:12	5:33:31	0:01:40	1 day 18:43:21
20000	1500	4800	1072	0:00:25	9:44:07	0:01:08	2 days 20:08:20
20000	2000	4800	1430	0:00:31	15:10:01	0:01:29	3 days 19:28:56
20000	2500	4800	1788	0:01:23	1 day 12:24:25	0:03:30	4 days 20:53:53
20000	3000	4800	2145	0:01:50	1 day 20:29:59	0:03:44	5 days 19:45:31
20000	3500	4800	2503	0:02:27	2 days 5:40:09	0:03:56	6 days 19:44:54
20000	4000	4800	2861	0:03:03	2 days 16:50:51	0:03:46	7 days 18:17:21
20000	4500	4800	3218	0:04:00	3 days 3:35:02	0:04:28	8 days 19:49:26
20000	5000	4800	3576	0:04:52	3 days 16:50:21	0:04:44	9 days 23:56:16

Logistic regression with heterogeneous data

N	F	Cores	Space(GB)	TWalltime	TCompute	CWalltime	CCompute
50000	2000	800	596	0:00:56	6:14:57	0:09:25	3 days 19:56:38
50000	2000	1600	1192	0:01:01	12:28:12	0:09:35	7 days 19:00:17
50000	2000	2400	1788	0:00:58	18:43:11	0:09:35	11 days 13:26:10
50000	2000	3200	2384	0:00:58	1 day 1:09:09	0:09:39	15 days 10:33:19
50000	2000	4000	2980	0:01:23	1 day 7:34:22	0:09:49	19 days 6:45:31
50000	2000	4800	3576	0:01:11	1 day 13:51:15	0:34:30	77 days 5:23:50
50000	2000	5600	4172	0:01:29	1 day 20:20:50	0:34:38	90 day 19:10:12
50000	2000	6400	4768	0:01:01	2 days 2:55:20	0:35:31	103 days 19:09:22
50000	2000	7200	5364	0:01:14	2 days 9:38:02	0:10:26	34 days 20:11:28
100000	1000	2000	1490	0:01:31	11:15:47	0:26:49	23 days 21:30:59
100000	1000	4000	2980	0:01:03	22:44:45	0:25:23	48 days 17:23:23
100000	1000	6000	4470	0:00:42	1 day 10:38:14	0:24:38	73 days 15:10:07
100000	1000	8000	5960	0:00:43	1 day 22:25:35	0:25:08	98 days 12:53:22
100000	1000	10000	7450	0:00:56	2 days 10:13:27	0:25:39	123 days 0:26:26
100000	1000	12000	8940	0:01:24	2 days 22:10:47	0:25:00	146 days 22:00:35
100000	1000	14000	10430	0:01:16	4 days 11:33:53	0:26:27	171 days 8:40:10
100000	1000	16000	11920	0:00:56	3 days 22:59:09	0:25:18	195 days 19:54:41
100000	1000	18000	13411	0:01:26	4 days 11:34:10	0:26:03	218 days 19:17:19
100000	1000	20000	14901	0:01:59	4 days 23:59:15	0:26:27	243 days 4:55:47

Lasso with heterogeneous data

N	F	Cores	Space(GB)	TWalltime	TCompute	CWalltime	CCompute
50000	200	800	59	0:00:12	0:01:45	0:00:37	0:04:55
50000	200	1600	119	0:00:02	0:03:31	0:00:47	0:10:56
50000	200	2400	178	0:00:02	0:05:14	0:01:14	0:17:50
50000	200	3200	238	0:00:00	0:07:00	0:01:22	0:25:24
50000	200	4000	298	0:00:04	0:09:00	0:01:36	0:33:49
50000	200	4800	357	0:00:11	0:10:25	0:01:57	0:43:29
50000	200	5600	417	0:00:10	0:12:09	0:02:07	0:55:47
50000	200	6400	476	0:00:07	0:13:48	0:02:19	1:04:51
50000	200	7200	536	0:00:09	0:15:31	0:02:39	1:19:22
50000	1000	800	298	0:00:04	0:33:28	0:05:20	2:58:02
50000	1000	1600	596	0:00:18	1:06:33	0:06:23	6:00:37
50000	1000	2400	894	0:00:25	1:39:50	0:08:28	9:04:25
50000	1000	3200	1192	0:00:09	2:12:14	0:08:34	12:07:04
50000	1000	4000	1490	0:00:08	2:46:27	0:09:52	15:13:18
50000	1000	4800	1788	0:00:21	3:24:38	0:13:28	18:11:34
50000	1000	5600	2086	0:00:10	3:50:29	0:14:55	21:25:49
50000	1000	6400	2384	0:00:06	4:26:31	0:16:11	1 day 0:27:56
50000	1000	7200	2682	0:00:11	4:56:57	0:17:11	1 day 3:34:19

SVM with homogeneous data

N	F	Cores	Space(GB)	TWalltime	TCompute	CWalltime	CCompute
50000	20	48	0	0:00:01	0:00:46	0:02:45	2:01:12
50000	20	96	0	0:00:01	0:01:32	0:02:47	4:03:05
50000	20	144	1	0:00:02	0:02:19	0:02:49	5:58:08
50000	20	192	1	0:00:02	0:03:06	0:02:45	7:56:14
50000	20	240	1	0:00:02	0:03:53	0:02:51	9:54:05
50000	50	48	0	0:00:03	0:01:23	0:05:14	3:44:06
50000	50	96	1	0:00:03	0:02:47	0:05:19	7:26:30
50000	50	144	2	0:00:03	0:04:11	0:05:25	11:07:51
50000	50	192	3	0:00:07	0:05:38	0:05:25	14:54:03
50000	50	240	4	0:00:03	0:07:00	0:05:25	18:26:10
50000	100	48	1	0:00:05	0:02:20	0:09:28	6:25:55
50000	100	96	3	0:00:05	0:04:40	0:09:56	12:49:20
50000	100	144	5	0:00:05	0:07:04	0:09:45	19:09:22
50000	100	192	7	0:00:06	0:09:25	0:09:53	1 day 1:27:47
50000	100	240	8	0:00:05	0:11:46	0:10:06	1 day 8:08:18

Star data

Cores	TWalltime	TCompute	CWalltime	CCompute
2500	0:01:06	11:35:25	0:24:39	31 days 19:59:13
3000	0:00:49	12:10:33	0:21:43	32 days 2:44:11
3500	0:00:50	12:17:27	0:17:01	30 days 7:56:19
4000	0:00:45	12:38:24	0:29:53	40 days 13:38:19